/\*

Name : X

Section : X

University Roll No. : X

1. Write a C Program to store N elements to an array and then send all positive elements of the array to the end without altering the original sequence.

\*/

#include<stdio.h>

void positive(int[], int); //Declaration of function

int main()

{

int i, n;

printf("Enter number of elements: ");

scanf("%d",&n);

int a[n]; //Declaration of an array

for(i=0; i<n; i++) //Taking inputs in array

{

printf("Enter element %d = ", i+1);

scanf("%d", &a[i]);

}

positive(a, n); //Calling function

for(i=0; i<n; i++) //Printing elements from array

printf("%d\t",a[i]);

return 0;

}

void positive(int a[], int n) //Defination of a function

{

int i, j, temp;

for(i=0; i<n; i++)

{

for(j=0; j<n-i-1; j++)

{

if(a[j]>=0 && a[j+1]<0)

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

}

}

\*\*\*\*\*OUTPUT\*\*\*\*\*

Enter number of elements: 5

Enter element 1 = -10

Enter element 2 = 9

Enter element 3 = -2

Enter element 4 = 12

Enter element 5 = 5

-10 -2 9 12 5

/\*

Name : X

Section : X

University Roll No. : X

2. Write a C program to create two linked lists positive and negative from a Original linked list, so that positive linked list contains all positive elements and negative linked list contains negative elements. Positive and negative linked lists should use the node of existing original linked list.

\*/

#include<stdio.h>

#include<stdlib.h>

typedef struct node //Declaration of structure of node

{

int data;

struct node \*next;

}NODE;

NODE\* insert(NODE\*, int); //Declaration of function for insertion

void traverse(NODE\*); //Declaration of function to display

void divide(NODE\*); //Declaration of function to separate the original list

int main()

{

int ch, ele;

char x='Y';

printf("Menu:-\n1. Insert in original\n2. Traverse positive and negative linked list\n3. Exit\n");

NODE \*start=NULL;

do

{

printf("Enter your choice: ");

scanf("%d", &ch);

switch (ch)

{

case 1: //For insertion

printf("Enter number: ");

scanf("%d", &ele);

start=insert(start, ele); //Calling insertion function

break;

case 2: //To separate the list

divide(start); //Calling separation function

break;

case 3: //To exit the code

x='N';

break;

default: //Invalid entry from the menu

printf("Invalid choice\n");

break;

}

} while (x=='Y');

return 0;

}

NODE\* insert(NODE\* start, int ele) //Defination of insertion function

{

NODE \*ptr=(NODE\*)malloc(sizeof(NODE)); //Creating new node

ptr->data=ele;

if (start==NULL)

{

ptr->next=NULL;

return ptr;

}

else

{

NODE \*t=start;

while(t->next!=NULL)

t=t->next;

t->next=ptr;

ptr->next=NULL;

return start;

}

}

void traverse(NODE\* start) //Defination of display function

{

if (start==NULL)

printf("No data available\n");

else

{

NODE \*t=start;

while(t->next!=NULL) //Printing elements of list

{

printf("%d\t", t->data);

t=t->next;

}

printf("%d\n", t->data);

}

}

void divide(NODE \*start) //Defination to separate the list

{

NODE \*pos=NULL,\*p=NULL,\*neg=NULL,\*n=NULL, \*t=NULL;

for(t=start; t!=NULL; t=t->next)

{

if(t->data>0)

{

if(pos==NULL)

pos=t;

else

p->next=t;

p=t;

}

else

{

if(neg==NULL)

neg=t;

else

n->next=t;

n=t;

}

}

p->next=NULL;

n->next=NULL;

printf("Positive Linked List:-\n");

traverse(pos); //Calling display function

printf("Negative Linked List:-\n");

traverse(neg); //Calling display function

}\*\*\*\*\*OUTPUT\*\*\*\*\*

Menu:-

1. Insert in original

2. Traverse positive and negative linked list

3. Exit

Enter your choice: 1

Enter number: 10

Enter your choice: 1

Enter number: 20

Enter your choice: 1

Enter number: -30

Enter your choice: 40

Invalid choice

Enter your choice: 1

Enter number: 40

Enter your choice: 1

Enter number: -50

Enter your choice: 2

Positive Linked List:-

10 20 40

Negative Linked List:-

-30 -50

Enter your choice: 3

/\*

Name : X

Section : X

University Roll No. : X

3. Write a C program to create a linked list P, and then write a ‘C’ function named split to create two linked lists Q & R from P So that Q contains all elements in odd positions of P and R contain the remaining elements. Finally print both linked lists i.e., Q and R.

\*/

#include<stdio.h>

#include<stdlib.h>

typedef struct node //Declaration of structure of node

{

int data;

struct node \*next;

}NODE;

NODE\* insert(NODE\*, int); //Declaration of function for insertion

void traverse(NODE\*); //Declaration of function to display

void divide(NODE\*); //Declaration of function to separate the original list

int main()

{

int ch, ele;

char x='Y';

printf("Menu:-\n1. Insert in original\n2. Traverse even and odd linked list\n3. Exit\n");

NODE \*start=NULL;

do

{

printf("Enter your choice: ");

scanf("%d", &ch);

switch (ch)

{

case 1: //For insertion

printf("Enter number: ");

scanf("%d", &ele);

start=insert(start, ele); //Calling insertion function

break;

case 2: //To separate the list

divide(start); //Calling separation function

break;

case 3: //To exit the code

x='N';

break;

default: //Invalid entry from the menu

printf("Invalid choice\n");

break;

}

} while (x=='Y');

return 0;

}

NODE\* insert(NODE\* start, int ele) //Defination of insertion function

{

NODE \*ptr=(NODE\*)malloc(sizeof(NODE)); //Creating new node

ptr->data=ele;

if (start==NULL)

{

ptr->next=NULL;

return ptr;

}

else

{

NODE \*t=start;

while(t->next!=NULL)

t=t->next;

t->next=ptr;

ptr->next=NULL;

return start;

}

}

void traverse(NODE\* start) //Defination of display function

{

if (start==NULL)

printf("No data available\n");

else

{

NODE \*t=start;

while(t->next!=NULL) //Printing elements of list

{

printf("%d\t", t->data);

t=t->next;

}

printf("%d\n", t->data);

}

}

void divide(NODE \*start) //Defination to separate the list

{

NODE \*even=NULL,\*e=NULL,\*odd=NULL,\*o=NULL, \*t=NULL;

int i=1;

for(t=start; t!=NULL; t=t->next)

{

if(i%2==0)

{

if(even==NULL)

even=t;

else

e->next=t;

e=t;

}

else

{

if(odd==NULL)

odd=t;

else

o->next=t;

o=t;

}

i++;

}

e->next=NULL;

o->next=NULL;

printf("Even Position Linked List:-\n");

traverse(even); //Calling display function

printf("Odd Position Linked List:-\n");

traverse(odd); //Calling display function

}\*\*\*\*\*OUTPUT\*\*\*\*\*

Menu:-

1. Insert in original

2. Traverse even and odd linked list

3. Exit

Enter your choice: 1

Enter number: 2

Enter your choice: 1

Enter number: 4

Enter your choice: 1

Enter number: 6

Enter your choice: 1

Enter number: 8

Enter your choice: 1

Enter number: 10

Enter your choice: 2

Even Position Linked List:-

4 8

Odd Position Linked List:-

2 6 10

Enter your choice: 3

/\*

Name : X

Section : X

University Roll No. : X

4. Write a C program to add of two polynomials of degree n, using linked   
p1 = first polynomial   
p2 = second polynomial   
Find out p3= p1+p2

\*/

#include <stdio.h>

#include<stdlib.h>

typedef struct node //Declaration of structure of node

{

int data, ex;

struct Node \*next;

}NODE;

NODE\* insert(NODE\*, int, int); //Declaration of function for insertion

NODE \*add(NODE\*, NODE\*); //Declaration of function to add two polynomials

void traverse(NODE\*); //Declaration of function to display

int main()

{

NODE \*list1=NULL, \*list2=NULL,\*list3=NULL;

int ch, ele, n, i, e;

char x='Y';

printf("Menu:-\n1. Insert in List 1\n2. Insert in List 2\n3. Add both lists\n4. Exit\n");

do

{

printf("Enter your choice: ");

scanf("%d", &ch);

switch (ch)

{

case 1: //For insertion in list 1

printf("Enter number of terms in list 1: ");

scanf("%d", &n);

for(i=1; i<=n; i++)

{

printf("Enter number & exponent:\n");

scanf("%d%d", &ele, &e);

list1=insert(list1, ele, e); //Calling insertion function

}

traverse(list1); //Calling display function

break;

case 2: //For insertion in list 2

printf("Enter number of terms in list 2: ");

scanf("%d", &n);

for(i=1; i<=n; i++)

{

printf("Enter number & exponent:\n");

scanf("%d%d", &ele, &e);

list2=insert(list2, ele, e); //Calling insertion function

}

traverse(list2); //Calling display function

break;

case 3: //To add polynomials

list3=add(list1,list2); //Calling function to adding

traverse(list3); //Calling display function

break;

case 4: //To exit the code

x='N';

break;

default: //Invalid entry from the menu

printf("Invalid choice\n");

break;

}

} while (x=='Y');

return 0;

}

NODE\* insert(NODE \*start, int ele, int e) //Defination of insertion function

{

NODE\* ptr=(NODE\*)malloc(sizeof(NODE)); //Creating new node

ptr->data=ele;

ptr->ex=e;

ptr->next=NULL;

if (start==NULL)

return ptr;

NODE \*t=start, \*p=NULL;

while (t!=NULL && t->ex<e)

{

p=t;

t=t->next;

}

if (p==NULL)

{

ptr->next=start;

return ptr;

}

else

{

p->next=ptr;

ptr->next=t;

return start;

}

}

NODE \*add(NODE \*l1, NODE \*l2) //Defination of adding polynomials

{

NODE \*l3=NULL;

while(l1!=NULL && l2!=NULL)

{

NODE \*ptr=(NODE\*)malloc(sizeof(NODE));

if(l1->ex==l2->ex)

{

ptr->data=l1->data+l2->data;

ptr->ex=l1->ex;

l1=l1->next;

l2=l2->next;

}

else if(l1->ex<l2->ex)

{

ptr->data=l1->data;

ptr->ex=l1->ex;

l1=l1->next;

}

else

{

ptr->data=l2->data;

ptr->ex=l2->ex;

l2=l2->next;

}

ptr->next=NULL;

if(l3==NULL)

l3=ptr;

else

{

ptr->next=l3;

l3=ptr;

}

}

while(l1!=NULL) //If list 1 having more elements than list 2

{

NODE \*ptr=(NODE\*)malloc(sizeof(NODE));

ptr->data=l1->data;

ptr->ex=l1->ex;

ptr->next=NULL;

if(l3==NULL)

l3=ptr;

else

{

ptr->next=l3;

l3=ptr;

}

l1=l1->next;

}

while(l2!=NULL) //If list 2 having more elements than list 1

{

NODE \*ptr=(NODE\*)malloc(sizeof(NODE));

ptr->data=l2->data;

ptr->ex=l2->ex;

ptr->next=NULL;

if(l3==NULL)

l3=ptr;

else

{

ptr->next=l3;

l3=ptr;

}

l2=l2->next;

}

return l3;

}

void traverse(NODE \*start) //Defination of display function

{

NODE \*t=start;

printf("Display:-\n");

while(t!=NULL) //Printing elements of list

{

printf("%dx%d\t",t->data,t->ex);

t=t->next;

}

printf("\n");

}\*\*\*\*\*OUTPUT\*\*\*\*\*

Menu:-

1. Insert in List 1

2. Insert in List 2

3. Add both lists

4. Exit

Enter your choice: 1

Enter number of terms in list 1: 4

Enter number & exponent:

10 4

Enter number & exponent:

7 3

Enter number & exponent:

6 1

Enter number & exponent:

9 0

Display:-

9x0 6x1 7x3 10x4

Enter your choice: 2

Enter number of terms in list 2: 3

Enter number & exponent:

12 3

Enter number & exponent:

4 2

Enter number & exponent:

8 0

Display:-

8x0 4x2 12x3

Enter your choice: 3

Display:-

10x4 19x3 4x2 6x1 17x0

/\*

Name : X

Section : X

University Roll No. : X

5. Write a C program to implement time sharing environment (using circular linked list) for N processes, where CPU allocates time slots of 10ns for given N processes, then print which process will be completed in how much time.

\*/

#include <stdio.h>

#include <stdlib.h>

typedef struct node //Declaration of structure of node

{

struct node \*next;

int info;

int sno;

} NODE;

NODE\* insert(NODE\*,int); //Declaration of function for insertion

void traverse(NODE\*); //Declaration of function to display

void delete(NODE\*\*); //Declaration of deletion of node

void taskprocess(NODE\*\*, int); //Declaration of CPU processing

int main()

{

NODE \*top=NULL;

int choice, x, ch=1;

printf("OPERATION YOU NEED TO PERFORM:\n1. INSERT\n2. PROCESS SCHEDULING\n3. DISPLAY\n4. EXIT\n");

do

{

printf("Enter your choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1: //For insertion

printf("Enter time: ");

scanf("%d", &x);

top=insert(top, x); //Calling insertion function

break;

case 2: //CPU processing

taskprocess(&top, 10); //Calling CPU processing

break;

case 3: //To display

traverse(top); //Calling display function

break;

case 4: //To exit the code

ch=0;

break;

default: //Invalid entry from the menu

printf("Invalid choice hence exit\n");

break;

}

} while(ch==1);

}

NODE\* insert(NODE \*start,int n) //Defination of insertion function

{

static int sno=0;

NODE \*ptr=(NODE\*)malloc(sizeof(NODE)); //Creating new node

ptr->sno=++sno;

ptr->info=n;

if(start==NULL)

ptr->next=ptr;

else

{

ptr->next=start->next;

start->next=ptr;

}

return ptr;

}

void traverse(NODE\* start) //Defination of display function

{

if (start==NULL)

printf("No data available\n");

else

{

NODE \*t=start;

printf("Time Taken\n");

while(t->next!=start) //Printing elements of list

{

printf("%d\t", t->info);

t=t->next;

}

printf("%d\n", t->info);

}

}

void delete(NODE \*\*p) //Defination of deleting a node

{

NODE \*q=\*p, \*r=NULL;

if(q->next==q)

{

free(q);

\*p=NULL;

}

else

{

r=q->next;

q->next=r->next;

free(r);

\*p=q;

}

}

void taskprocess(NODE \*\*p, int timee) //Defination of CPU processing

{

int c=0, y;

NODE \*q=\*p;

while(q)

{

y=(++c)\*timee;

NODE \*f=q->next;

f->info-=timee;

if((f->info)<=0)

{

printf("Process-%d is completed in %d turn within %d unit time\n", f->sno, c, y);

delete(&q);

}

else q=q->next;

}

}

\*\*\*\*\*OUTPUT\*\*\*\*\*

OPERATION YOU NEED TO PERFORM:

1. INSERT

2. PROCESS SCHEDULING

3. DISPLAY

4. EXIT

Enter your choice: 1

Enter time: 20

Enter your choice: 1

Enter time: 30

Enter your choice: 1

Enter time: 50

Enter your choice: 1

Enter time: 40

Enter your choice: 3

Time Taken

40 20 30 50

Enter your choice: 2

Process-1 is completed in 5 turn within 50 unit time

Process-2 is completed in 9 turn within 90 unit time

Process-4 is completed in 13 turn within 130 unit time

Process-3 is completed in 14 turn within 140 unit time

Enter your choice: 4

/\*

Name : X

Section : X

University Roll No. : X

6. Write a C program to create a double linked list by inserting nodes in such a way that the resultant linked list remains in ascending order.(do not use any sorting technique).

\*/

#include<stdio.h>

#include<stdlib.h>

typedef struct node //Declaration of structure of node

{

int data;

struct node \*next;

struct node \*pre;

} NODE;

NODE\* insert(NODE\*, int); //Declaration of function for insertion

void traverse(NODE\*); //Declaration of function to display

int main()

{

int ch, ele, n;

char x='Y';

printf("Menu:-\n1. Insert\n2. Traverse\n3. Exit\n");

NODE \*start=NULL;

do

{

printf("Enter your choice: ");

scanf("%d", &ch);

switch (ch)

{

case 1: //For insertion

printf("Enter number: ");

scanf("%d", &ele);

start=insert(start, ele); //Calling insertion function

break;

case 2: //To display

traverse(start); //Calling display function

break;

case 3: //To exit the code

x='N';

break;

default: //Invalid entry from the menu

printf("Invalid choice\n");

break;

}

} while (x=='Y');

return 0;

}

NODE\* insert(NODE\* start, int ele) //Defination of insertion function

{

NODE \*ptr=(NODE\*)malloc(sizeof(NODE)); //Creating new node

ptr->data=ele;

ptr->pre=NULL;

ptr->next=NULL;

if (start==NULL)

return ptr;

else if(start->data>ele)

{

ptr->next=start;

return ptr;

}

else

{

NODE \*t=start;

while(t->next!=NULL && t->next->data<ele)

t=t->next;

ptr->next=t->next;

if (t->next!=NULL)

t->next->pre=ptr;

t->next=ptr;

ptr->pre=t;

return start;

}

}

void traverse(NODE\* start) //Defination of display function

{

if (start==NULL)

printf("No data available\n");

else

{

NODE \*t=start;

while(t->next!=NULL) //Printing elements of list

{

printf("%d\t", t->data);

t=t->next;

}

printf("%d\n", t->data);

}

}

\*\*\*\*\*OUTPUT\*\*\*\*\*

Menu:-

1. Insert

2. Traverse

3. Exit

Enter your choice: 1

Enter number: 10

Enter your choice: 1

Enter number: 20

Enter your choice: 1

Enter number: 30

Enter your choice: 1

Enter number: 5

Enter your choice: 1

Enter number: 15

Enter your choice: 1

Enter number: 25

Enter your choice: 2

5 10 15 20 25 30

Enter your choice: 3

/\*

Name : X

Section : X

University Roll No. : X

7. Write a C program to create a binary search tree and perform following operations:   
 1) Search a particular key.   
 2) Delete a node from the tree.   
 3) Find total number of leaf nodes   
 4) Find height of a binary search tree   
 5) Count total numbers of nodes from right hand side of root node

\*/

#include<stdio.h>

#include<stdlib.h>

typedef struct node //Declaration of structure of node

{

struct node \*left;

int info;

struct node \*right;

} NODE;

NODE \*insert(NODE\*, int); //Declaration of function to inserting an element in tree

void search(NODE\*, int); //Declaraton of function to searching an element

NODE\* delete(NODE\*, int); //Declaration of function to deleting an element from a tree

int count\_leaf\_nodes(NODE\*); //Declaration of function to count leaf nodes

int count\_height(NODE\*); //Declaration of function to count height of tree

int count\_nodes(NODE\*); //Declaration of function to count nodes

int main()

{

NODE \*tree=NULL;

int x, ch, total\_leaf\_nodes, total\_right\_nodes, height;

char y='Y';

printf("0. Insert\n1. Search key\n2. Delete\n3. Count leaf nodes\n4. Height\n5. Nodes from right hand\n6. Exit\n");

do

{

printf("Enter your choice: ");

scanf("%d", &ch);

switch (ch)

{

case 0: //For insert

printf("Enter an element: ");

scanf("%d", &x);

tree=insert(tree, x); //Calling insertion function

break;

case 1: //For searching

printf("Enter an element: ");

scanf("%d", &x);

search(tree, x); //Calling search function

break;

case 2: //To delete a node

printf("Enter an element: ");

scanf("%d", &x);

tree=delete(tree, x); //Calling delete function

break;

case 3: //To count leaf nodes

total\_leaf\_nodes=count\_leaf\_nodes(tree); //Calling function to count leaf nodes

printf("Total number of leaf nodes: %d\n", total\_leaf\_nodes);

break;

case 4: //To count height

height=count\_height(tree); //Calling function to count height

printf("Height of tree: %d\n", height);

break;

case 5: //To count total nodes

total\_right\_nodes=count\_nodes(tree->right); //Calling function to count nodes

printf("Total number of nodes: %d\n", total\_right\_nodes);

break;

case 6: //To exit the code

y='n';

break;

default: //Invalid entry from the menu

printf("Invalid entry\n");

break;

}

} while (y=='Y');

return 0;

}

NODE \*insert(NODE \*tree, int ele) //Defination of insertion function

{

if (tree==NULL)

{

tree=(NODE\*)malloc(sizeof(NODE));

tree->left=NULL;

tree->info=ele;

tree->right=NULL;

}

else if (ele>(tree->info))

tree->right=insert(tree->right, ele);

else

tree->left=insert(tree->left, ele);

return tree;

}

void search(NODE \*tree, int ele) //Defination of search function

{

if (tree==NULL)

printf("Not found\n");

else if (tree->info==ele)

printf("Element found\n");

else if (ele>tree->info)

search(tree->right, ele);

else

search(tree->left, ele);

}

NODE\* delete(NODE\* tree, int ele) //Defination of delete function

{

if (tree==NULL)

return NULL;

if (ele<tree->info)

tree->left=delete(tree->left, ele);

else if (ele>tree->info)

tree->right=delete(tree->right, ele);

else

{

if (tree->left==NULL && tree->right==NULL)

return NULL;

else if (tree->left==NULL)

return tree->right;

else if (tree->right==NULL)

return tree->left;

else

{

NODE\* p=NULL;

for(p=tree; p->left!=NULL; p=p->right);

tree->info=p->info;

tree->right=delete(tree->right, tree->info);

}

}

}

int count\_leaf\_nodes(NODE \*tree) //Defination of count leaf node function

{

if(tree==NULL)

return 0;

else if ((tree->left)==NULL && (tree->right)==NULL)

return 1;

else

return count\_leaf\_nodes(tree->left)+count\_leaf\_nodes(tree->right);

}

int count\_height(NODE \*tree) //Defination of count height function

{

if(tree==NULL)

return 0;

else

{

int l, r;

l=count\_height(tree->left);

r=count\_height(tree->right);

return (l>r)?(l+1):(r+1);

}

}

int count\_nodes(NODE \*tree) //Defination of count nodes function

{

if(tree==NULL)

return 0;

else

return 1+count\_nodes(tree->left)+count\_nodes(tree->right);

}

\*\*\*\*\*OUTPUT\*\*\*\*\*

0. Insert

1. Search key

2. Delete

3. Count leaf nodes

4. Height

5. Nodes from right hand

6. Exit

Enter your choice: 0

Enter an element: 10

Enter your choice: 0

Enter an element: 5

Enter your choice: 0

Enter an element: 15

Enter your choice: 0

Enter an element: 8

Enter your choice: 0

Enter an element: 20

Enter your choice: 1

Enter an element: 15

Element found

Enter your choice: 2

Enter an element: 15

Enter your choice: 3

Total number of leaf nodes: 2

Enter your choice: 4

Height of tree: 4

Enter your choice: 5

Total number of nodes: 1

Enter your choice: 5

/\*

Name : X

Section : X

University Roll No. : X

8. Write a C program to sort an unsorted sequence of strings given by user in an array, using Merge sort technique.

\*/

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

void merge(char[][50], int, int, int); //Declaration of merging function

void mergeSort(char [][50], int, int); //Declaration of merge sort function

void display(char [][50], int); //Declaration of display function

int main()

{

int n,i;

printf("Enter number of string: ");

scanf("%d", &n);

char arr[n][50];

for(i=0; i<n; i++) //Taking inputs of array

{

printf("Enter string: ");

scanf("%s", arr[i]);

}

printf("Given array: \n");

display(arr, n); //Calling display function

mergeSort(arr, 0, n-1); //Calling merge sort function

printf("Sorted array: \n");

display(arr, n); //Calling display function

return 0;

}

void merge(char arr[][50], int l, int m, int r) //Defination of merge function

{

int i, j, k, n1=m-l+1, n2=r-m;

char L[n1][50], R[n2][50];

for (i=0; i<n1; i++)

strcpy(L[i], arr[l+i]);

for (j=0; j<n2; j++)

strcpy(R[j], arr[m+1+j]);

i=0, j=0, k=l;

while (i<n1 && j<n2)

{

if (strcmp(L[i], R[j])<=0)

strcpy(arr[k++] ,L[i++]);

else

strcpy(arr[k++], R[j++]);

}

while (i<n1)

strcpy(arr[k++], L[i++]);

while (j<n2)

strcpy(arr[k++], R[j++]);

}

void mergeSort(char arr[][50], int l, int r) //Defination of merge sort function

{

if (l<r)

{

int m=l+(r-l)/2;

mergeSort(arr, l, m);

mergeSort(arr, m+1, r);

merge(arr, l, m, r);

}

}

void display(char A[][50], int size) //Defination of display function

{

int i;

for (i=0; i<size; i++) //Printing strings from array

printf("%s\t", A[i]);

printf("\n");

}

\*\*\*\*\*OUTPUT\*\*\*\*\*

Enter number of strings: 5

Enter string: Hello

Enter string: World

Enter string: Computer

Enter string: Is

Enter string: Here

Sorted array:

Computer Hello Here Is World

/\*

Name : X

Section : X

University Roll No. : X

9. Write a C program to implement linked representation of a graph in memory using array of pointers.

\*/

#include <stdio.h>

#include <stdlib.h>

struct AdjListNode //Structure of adjacent list node

{

int dest;

struct AdjListNode\* next;

};

struct AdjList //Structure of adjacent list

{

struct AdjListNode\* head;

};

struct Graph //Structure of graph

{

int V;

struct AdjList\* array;

};

struct AdjListNode\* newAdjListNode(int); //Declaration of creating adjacent list node

struct Graph\* createGraph(int); //Declaration of creating graph

void addEdge(struct Graph\*, int, int); //Declaration of edges function

void printGraph(struct Graph\*); //Declaration to display function

int main()

{

int V=5;

struct Graph\* graph=createGraph(V);

addEdge(graph, 0, 1); //Calling egde function with parameters

addEdge(graph, 0, 4); //Calling egde function with parameters

addEdge(graph, 1, 2); //Calling egde function with parameters

addEdge(graph, 1, 3); //Calling egde function with parameters

addEdge(graph, 1, 4); //Calling egde function with parameters

addEdge(graph, 2, 3); //Calling egde function with parameters

addEdge(graph, 3, 4); //Calling egde function with parameters

printGraph(graph); //Calling display function

return 0;

}

struct AdjListNode\* newAdjListNode(int dest) //Defination of creating adjacent list node

{

struct AdjListNode\* newNode=(struct AdjListNode\*)malloc(sizeof(struct AdjListNode));

newNode->dest=dest;

newNode->next=NULL;

return newNode;

}

struct Graph\* createGraph(int V) //Defination of creating graph

{

struct Graph\* graph=(struct Graph\*)malloc(sizeof(struct Graph));

graph->V=V;

graph->array=(struct AdjList\*)malloc(V\*sizeof(struct AdjList));

int i;

for (i=0; i<V; ++i)

graph->array[i].head = NULL;

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest) //Defination of edges function

{

struct AdjListNode\* check=NULL;

struct AdjListNode\* newNode=newAdjListNode(dest);

if (graph->array[src].head==NULL)

{

newNode->next=graph->array[src].head;

graph->array[src].head=newNode;

}

else

{

check=graph->array[src].head;

while (check->next != NULL)

check=check->next;

check->next=newNode;

}

newNode=newAdjListNode(src);

if (graph->array[dest].head==NULL)

{

newNode->next=graph->array[dest].head;

graph->array[dest].head=newNode;

}

else

{

check=graph->array[dest].head;

while (check->next!=NULL)

check=check->next;

check->next=newNode;

}

}

void printGraph(struct Graph\* graph) //Defination to display function

{

int v;

for (v=0; v<graph->V; ++v)

{

struct AdjListNode \*p=graph->array[v].head;

printf("Adjacency list of vertex %d\n head ", v);

while (p)

{

printf("-> %d", p->dest);

p=p->next;

}

printf("\n");

}

}

\*\*\*\*\*OUTPUT\*\*\*\*\*

Adjacency list of vertex 0

head -> 1-> 4

Adjacency list of vertex 1

head -> 0-> 2-> 3-> 4

Adjacency list of vertex 2

head -> 1-> 3

Adjacency list of vertex 3

head -> 1-> 2-> 4

Adjacency list of vertex 4

head -> 0-> 1-> 3

/\*

Name : X

Section : X

University Roll No. : X

10. Write a C program to implement DFS.

\*/

#include<stdio.h>

void DFS(int G[10][10],int visited[10],int n,int); //Declaration of DFS

void main()

{

int G[10][10], visited[10], n, i, j;

printf("Enter number of vertices:"); //Taking number of vertices

scanf("%d",&n);

printf("Enter adjacency matrix of the graph:\n");

for(i=0;i<n;i++) //Taking adjacency matrix

for(j=0;j<n;j++)

scanf("%d",&G[i][j]);

for(i=0;i<n;i++)

visited[i]=0;

DFS(G,visited,n,0); //Calling DFS

}

void DFS(int G[10][10],int visited[10],int n,int i) //Defination of DFS

{

int j;

printf("\n%d",i); //Print path

visited[i]=1;

for(j=0;j<n;j++)

if(!visited[j]&&G[i][j]==1)

DFS(G,visited,n,j); //Recurrsion of DFS

}

\*\*\*\*\*OUTPUT\*\*\*\*\*

Enter number of vertices:4

Enter adjacency matrix of the graph:

1 0 0 1

0 1 1 1

0 1 1 0

1 1 0 1

0

3

1

2

/\*

Name : X

Section : X

University Roll No. : X

11. Write a C Program to implement Kruskal’s algorithm.

\*/

#include <stdio.h>

#include <stdlib.h>

int find(int [],int); //Declaration of find function

int uni(int [],int, int); //Declaration of uni function

void main()

{

int i, j, k, a, b, u, v, n, ne = 1;

int min, mincost = 0, cost[9][9], parent[9]={0};

printf("Implementation of Kruskal's Algorithm\n");

printf("Enter the no. of vertices:"); //Taking number of vertices

scanf("%d", & n);

printf("Enter the cost adjacency matrix:\n"); //Taking cost of nodes

for (i = 1; i <= n; i++)

{

for (j = 1; j <= n; j++)

{

scanf("%d", & cost[i][j]);

if (cost[i][j] == 0)

cost[i][j] = 999;

}

}

printf("The edges of Minimum Cost Spanning Tree are\n");

while (ne < n)

{

for (i = 1, min = 999; i <= n; i++)

{

for (j = 1; j <= n; j++)

{

if (cost[i][j] < min)

{

min = cost[i][j];

a = u = i;

b = v = j;

}

}

}

u = find(parent,u); //Calling find function

v = find(parent,v); //Calling uni function

if (uni(parent,u, v))

{

printf("%d edge (%d,%d) =%d\n", ne++, a, b, min);

mincost += min;

}

cost[a][b] = cost[b][a] = 999;

}

printf("Minimum cost = %d\n", mincost); //Printing minimum cost

}

int find(int parent[],int i) //Defination of find function

{

while (parent[i])

i = parent[i];

return i;

}

int uni(int parent [],int i, int j) //Defination of uni function

{

if (i != j)

{

parent[j] = i;

return 1;

}

return 0;

}

\*\*\*\*\*OUTPUT\*\*\*\*\*

Implementation of Kruskal's Algorithm

Enter the no. of vertices:4

Enter the cost adjacency matrix:

0 1 0 0

2 0 2 1

3 0 0 3

0 0 0 0

The edges of Minimum Cost Spanning Tree are

1 edge (1,2) =1

2 edge (2,4) =1

3 edge (2,3) =2

Minimum cost = 4